Running Qiskit on ROCm Platform

Yujiang Bi[1,2,*], *Shun* Xu[3], and *Yunheng* Ma[4]

[1]Institute of High Energy Physics, CAS, 100049 Beijing, China

[2]Tianfu Cosmic Ray Research Center, 610213 Chengdu, China

[3]Computer Network Information Center, CAS, 100083 Beijing, China

[4]Beijing Academy of Quantum Information Sciences, 100193 Beijing, China

Abstract. Qiskit is one of the common quantum computing frameworks and and the qiskit-aer package can accelerating quantum circuit simulation using NVIDIA GPU with the help of THRUST. AMD ROCm framework similar to CUDA, a heterogeneous computing framework supporting both the NVIDIA and AMD GPUs provides the possibility to porting Qiskit/Qiskit-Aer from CUDA platform to its own. We present the porting progress of Qiskit/Qiskit-Aer and preliminary performance test on both NVIDA and AMD GPUs. Our results show that Qiskit/Qiskit-Aer cand work well on AMD GPUs with the help of ROCm/HIP, and has comparable performance on AMD platform.

1 Introduction

Quantum computing is one of the most promising directions of scientific computing and has showed many potential applications in quantum chemistry, bio-molecules, materials physics, drug development and other fields despite that we are and will be in the NISQ era[1] for a long time before the fault-tolerant quantum computing.

Digital quantum simulator, i.e. a routine using electronic computers to simulate processes of quantum computing, can verify the correctness of physical quantum computers to quickly develop and debug quantum algorithms.

There are many quantum computing development frameworks including own quantum simulator, and Qiskit[2] is one of the most commonly used quantum algorithm development frameworks. It's an open source quantum computing software package developed by IBM, including Qiskit-Aer quantum simulator, qiskit-aqua quantum application algorithm, etc. Qiskit-Aer provides different types of quantum simulators for simulating quantum circuit computation processes on classical computers, which is one of the core modules in Qiskit. Some simulators can accelerate the computation using NVIDIA GPUs based on CUDA[3, 4] and THRUST[5] library.

Qiskit and Qiskit-Aer can works well with NVIDIA GPUs, while has difficulty in cooperation with AMD GPUs. Since 2016, AMD proposed a heterogeneous computing framework named

*e-mail: biyujiang@ihep.ac.cn

HIP/ROCm[6, 7] similar to CUDA, and provides tools like hipify[8] to help developers porting their CUDA software to ROCm platform. Fig.1 shows the architecture of HIP/ROCm.

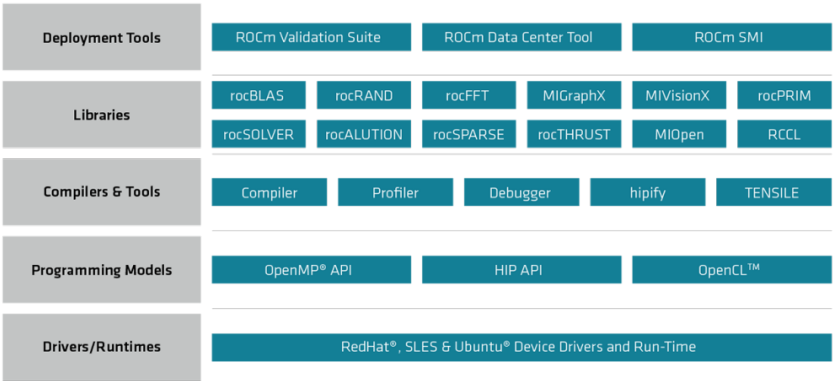

Figure 1. The architecture of HIP/ROCm platform, including driver/runtimes, programming models like OpenMP and OpenCL, compilers & tools, common libraries.

Up to now, many CUDA software or applications have been ported to ROCm platform, such as Tensorflow, PyTorch and QUDA[9, 10]. In this proceeding, we will present our effort on porting qiskit-aer to ROCm/HIP platform and the performance test on AMD GPUs.

2 Porting Progress

One of the challenges of porting was the level of THRUST support on HIP/ROCm. In addition, HIP/ROCm uses Clang as its backend, and some programs that can be compiled under CUDA may not necessarily be compiled under ROCm platform.

The code porting of Qiskit-Aer is mainly divided into three stages: compilation environment, Qiskit-Aer and THRUST library. In this section, we will overview our porting process briefly.

2.1 Code Portion

Qiskit-aer uses CMake as its compiling toolkit, so we added ROCm/HIP support in the cmake file via adding ROCm backend analog CUDA backend. Also we modified the CMake configuration to enable AVX2 support.

Qiskit-aer's GPU code is relatively centralized and mainly thrust code, so it is relatively easy to port. We used the recommended tolls hipify-perl and hipify-cmakefile to find what to be altered.

In order to unify the CUDA and ROCM frameworks, we have redefined some types and functions, such as cudaSuccess or hipSuccess redefined as *aerSuccess*, etc., and enable the different definitions through the *AER_THRUST_CUDA* or *AER_THRUST_ROCm* macros depending the compiling platform and backend. Some definitions are as following:

```
#ifdef AER_THRUST_ROCm
#include <hip/hip_runtime.h>
...
#define aerSuccess hipSuccess
#define aerError_t hipError_t
...
#endif
```

All the macros are defined in the header file *src/misc/gpu_macros.hpp*.

Qiskit-Aer uses THRUST for GPU-accelerated computation, which is also one of the difficulties in porting. THRUST relies heavily OpenMP instructions to accelerate computation, and provides a backend based on OpenMP, while HIP/ROCm has some problems in compiling a mix of OpenMP and GPU codes. In many cases, one can only use the *_OPENMP* macro to let the compiler determine whether to enable or disable OpenMP, such as replacing the OpenMP version of the *thrust::fill_n(thrust::omp::par,...)* function call with the serial version of *thrust::fill_n(...)*.

2.2 Compilation and Installation

We use GCC 7 and ROCm 5 to compiling Qiskit-Aer. There are some problems using conan to compile, so conan is disabled and "Unix Makefiles" for cmake generator is used. To run qiskit on multiple node, MPI support is enabled while GDR function is disabled due to compatibility issues. Following the development guide,

```
$ export DISABLE_CONAN=ON
$ export CMAKE_GENERATOR='Unix Makefiles'
$ export QISKIT_AER_PACKAGE_NAME=qiskit-aer-gpu
$ CC=hipcc CXX=hipcc LDFLAGS="-L/opt/base/lib -L/opt/base/lib64" python3
    setup.py bdist_wheel --build-type=RelWithDebInfo -- \
 -DAER_THRUST_BACKEND=ROCm -DAER_MPI=True -L -DAER_DISABLE_GDR=Tsrue -
    DBUILD_TESTS=False -- -j1 VERBOSE=1
```

After compiling about 3 hours in our environment, a package like *qiskit_aer_gpu-xxx-linux_x86_64.whl* will be generated under *dist* directory if no errors occur. We checked the GPU support with the following commands after installing the package manually:

```
from qiskit import Aer
print(Aer.backends())
[...,AerSimulator('aer_simulator_statevector_gpu'),...]
```

The Aer module contains *aer_simulator_statevector_gpu* and other GPU-support backends, which means that qiskit-aer has recognized AMD GPU in our environment.

The ROCm-compatable qiskit of v0.11.1 has been uploaded to github in case of any interest, and we will keep it up to the upstream if possible.

2.3 Correctness Validation

We use a simple quantum fourier transform (QFT) circuit to validating the correctness. Specifically, we choose the *aer_simulator_statevector_gpu* backend for validation, and perform a standard QFT on the initial state $|01\rangle$, followed by an inverse-QFT, then measure the final state, which turns out to be $|01\rangle$ as expected. The statevectors before and after QFT and inverse-QFT are the same, indicating that the calculation of our QFT circuit is correct.

3 Performance Test

For simplicity, we used a modified QFT workload to evaluate performance of qiskit simulators on NVIDIA and AMD GPUs. To make the results comparable, we chose products similar in performance and similar software environment.

3.1 Testbed Setup

The hardware setup and software we used for testing for AMD and NVIDIA platform are listed in table 1 and table 2:

Table 1. Hardware comparison of AMD and NVIDIA platform.

	AMD	NVIDIA
CPU	1 C86 7185 (32-Core)	2*Intel(R) Xeon(R) Gold 6240 (18-core)
RAM	8*16GB DDR4 2666MHz	12*32G DDR4 2933MHz
Accelerator	MI60, 16GB	V100, 32GB
Network	Infiniband 200Gb/s 4 vports	Infiniband 200Gb/s 2 vports

Table 2. Software environment comparison of AMD and NVIDIA platform.

	AMD	NVIDIA
Qiskit Aer	0.11.1	0.11.1
Kernel	3.10.0-957	3.10.0-1160.71.1
CentOS	7.6.1810	7.9.2009
OpenMPI	4.0.2a	4.1.4
UCX	1.6.0	1.13.1
Toolkit	ROCm 5	CUDA 11.7
GCC	7.3.1	9.3.1
CMake	3.24.2	3.24.2
Python	3.11.4	3.11.4

We tried to keep all software consistent, but desipte that, some fundamental libraries and kernel are unchangeable. For the following tests, we chosed the *aer_simulator_statevector_gpu* for evaluate the performance of qiskit simulation.

3.2 NVIDIA Platform

To evaluate the performance of AMD GPUs, we performed some benchmark on our NVIDIA V100 clusters, the hardware of which is shown in Tab. 1. Basically, we ran the QFT routine up to 5 times for various qubits with different *block_qubits*, which is a parameter of Qiskit-Aer. We used up to 4 nodes or 32 V100 GPUs, and the results are shown in Fig.2.

Strong scaling or scalability is widely used to indicate the ability of software to deliver greater computational power when the amount of resources is increased. From Fig.2, we could tell that:

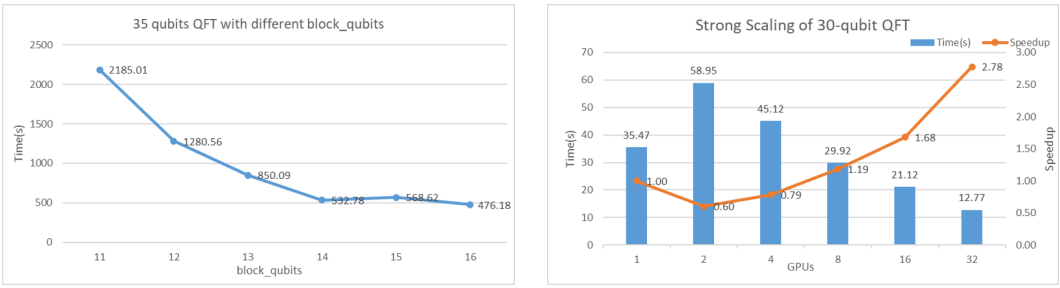

Figure 2. The performance of QFT on NVIDIA Telsa V100. Left: time cost for 35 qubits with different block_qubits. Right: strong scaling of 30-qubit QFT.

- The parameter *block_qutibs* significantly affects the simulation performance. The larger the value is, the higher the computational performance is.
- QFT on NVIDIA GPU has a good behavior of strong scaling.

3.3 AMD Platform

We performed the same benchmark on AMD platform, and the numerical results for 26-qubit QFT are shown in Tab. 3. There are some drawbacks for AMD platform, such as the lower

Table 3. QFT performance on AMD and NVIDIA Platform in seconds.

Platform	1 CPU Core	32 CPU Cores	1 GPU	GPU/(1 CPU Cores)	GPU/(32 CPU Cores)
AMD	108.34s	4.33s	1.59s	≈ 65	≈ 2.7
NVIDIA	58.90s	4.81s	1.86s	≈ 32	≈ 2.6

memory frequency and smaller video memory. Nevertheless, the performance on AMD GPU is quite impressive or even better than that on NVIDIA GPU, showing that AMD GPU and ROCm architecture are quite suitable for such workloads.

3.4 Large Scale Testing

On AMD platform, we performed the strong scaling test of 30-qubit QFT with $block\_qubits = 16$ up to 64 nodes or 256 GPUs. The numerical result on AMD/NVIDIA platform is shown in table 4 and the tendency on AMD platform is shown in Fig 3.

Table 4. QFT performance on AMD and NVIDIA Platform in seconds.

N(GPU)	1	2	4	8	16	32	64	96	128	192	256
AMD	36.33	51.31	48.72	31.49	20.05	14.75	6.98	6.64	4.04	3.41	2.33
NVIDIA	35.47 s	58.95	45.12	28.92	21.12	12.77					

From Tab. 4 and Fig. 3, we can find that:

- The performance and efficiency of QFT under AMD/NVIDIA platform are comparable.

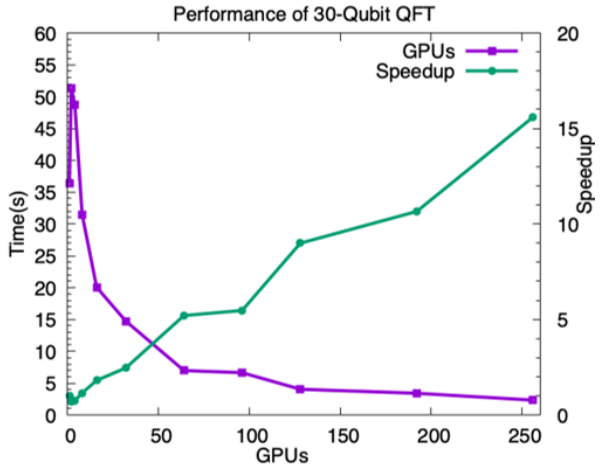

Figure 3. The strong scaling of performance on AMD Platform. The purple line shows the consumed time in second, and the green line shows the speedup with respect to a single GPU.

- Inside one node, the performance of one AMD/NVIDIA GPU is the best.
- The execution time decreases as the number of nodes increases.
- The efficiency under 16 nodes or 64 AMD GPUs is the best for 30-qubit QFT by virtue of experience.

4 Result and Outlook

The test above shows that qiskit simulators have good performance on AMD GPUs, which indicates qiskit-aer could work well with ROCm platform.

The result was exciting and showed the capacity of AMD GPUs on qiskit simualtion, though the benchmark workload is very naive and has no special meanings alone. Practical algorithms in NISQ era such as QAOA[11], VQE[12, 13] and QGAN[14] are more useful in High Energy Physics. In addition, noise simulation should also be taken into account in real research.

Next, we shall assess performance of simulating various practical algorithms with noise in consideration on AMD GPUs with latest ROCm framework. Besides, our AMD GPU for testing is kind of outdated, and it's necessary to try performance testing on new GPU devices.

5 Summary

Qikist is one of the most common quantum computing developing frameworks, and can make use of NVIDIA GPUs to accelerating digital quantum circuit simulations.

In this work, qiskit-aer was successfully ported from CUDA platform to AMD ROCm platform. Simple benchmark via a customised QFT workload showed that AMD GPUs are suited for Qiskit's analog computing scenarios, and the performance on our accelerator is comparable to that of NVIDIA Tesla V100.

Our result is very preliminary and more tests on different complicated quantum algorithms such as QAOA, VQE and QGAN are needed for evaluating performance of AMD GPUs on Qiskit simulation.

Acknowledgement

This work was supported by **Youth Innovation Promotion Association of CAS (No. 203013)** and **the GHfund A (No. ghfund202202014404)**.

References

[1] J. Preskill, Quantum **2**, 79 (2018)

[2] Qiskit contributors, *Qiskit: An open-source framework for quantum computing* (2023)

[3] NVIDIA, P. Vingelmann, F.H. Fitzek, *Cuda, release: 10.2.89* (2020), https://developer.nvidia.com/cuda-toolkit

[4] S. Cook, *CUDA Programming: A Developers Guide to Parallel Computing with GPUs*, 1st edn. (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012), ISBN 9780124159334

[5] N. Bell, J. Hoberock, *Thrust: A Productivity-Oriented Library for CUDA*, in *Thrust: A Productivity-Oriented Library for CUDA* (2012), https://api.semanticscholar.org/CorpusID:60775233

[6] J. Wu, A. Belevich, E. Bendersky, M. Heffernan, C. Leary, J. Pienaar, B. Roune, R. Springer, X. Weng, R. Hundt, *Gpucc: An Open-Source GPGPU Compiler*, in *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Association for Computing Machinery, New York, NY, USA, 2016), CGO '16, p. 105116, ISBN 9781450337786, https://doi.org/10.1145/2854038.2854041

[7] *ROCm*, https://rocmdocs.amd.com/en/latest/index.html

[8] *hipfiy*, https://github.com/ROCm-Developer-Tools/HIPIFY

[9] M. Clark, R. Babich, K. Barros, R. Brower, C. Rebbi, Computer Physics Communications **181**, 1517 (2010)

[10] Y. Bi, Y. Xiao, W. Guo, M. Gong, P. Sun, S. Xu, Y. bo Yang, EPJ Web Conf. **245**, 09008 (2020)

[11] E. Farhi, J. Goldstone, S. Gutmann, *A quantum approximate optimization algorithm* (2014), 1411.4028

[12] A. Peruzzo, J. McClean, P. Shadbolt, M.H. Yung, X.Q. Zhou, P.J. Love, A. Aspuru-Guzik, J.L. O'Brien, Nature Communications **5**, 4213 (2014), 1304.3061

[13] K. Bharti, A. Cervera-Lierta, T.H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J.S. Kottmann, T. Menke et al., Rev. Mod. Phys. **94**, 015004 (2022)

[14] P. Wang, D. Wang, Y. Ji, X. Xie, H. Song, X. Liu, Y. Lyu, Y. Xie, *Qgan: Quantized generative adversarial networks* (2019), 1901.08263